Deadline Assignment and Tardiness Control for Real-Time Data Services

Yan Zhou, Kyoung-Don Kang
Department of Computer Science
State University of New York at Binghamton
{yzhou,kang}@cs.binghamton.edu

Abstract

It is challenging to support the timeliness of real-time data service requests in data-intensive real-time applications such as online auction or stock trading, while maintaining the freshness of temporal data that capture the current real-world status. Although deadline-aware real-time scheduling would significantly enhance the timeliness of data services, it is not clear how to assign explicit feasible deadlines to data service requests in an open environment. To address the problem, we design a new deadline assignment scheme to derive feasible deadlines for real-time data service requests considering their individual data needs. Further, we develop a systematic closed-loop approach to supporting the desired tardiness—the actual service delay to deadline ratio—of real-time data services even in the presence of dynamic workloads. We choose the tardiness metric due to its expressiveness compared to the deadline miss ratio and utilization that saturate at 0 and 1 when the system is underutilized or overloaded, respectively. The performance evaluation results acquired in our real-time stock trading testbed show that the desired average/transient tardiness is closely supported. Consequently, the deadline miss ratio is significantly reduced compared to a state-of-art database system with a real-time scheduling extension.

1 Introduction

In a number of data-intensive real-time applications, such as stock trading, online auction, or traffic control, it is critical to process service requests within their deadlines using fresh temporal data that represent the current real-world status. However, it is challenging to support the desired timeliness in real-time data services, since database workloads may vary significantly, for example, due to the varying market status. At the same time, the database system has to continuously update temporal data.

It is known that deadline-aware real-time scheduling, such as the earliest deadline first (EDF) scheduling algorithm [14], has significant impacts on the performance of real-time data management [21]. Therefore, assigning feasible deadlines to data service requests is essential. If the assigned deadlines are too tight, a large number of deadline violations may occur, incurring potential instability of the system. On the other hand, too loose deadlines may not be able to support timing constraints specific to the application of interest such as e-commerce. Despite the importance, deadline assignment for real-time data service requests has received relatively little attention. Most existing work on real-time data management assumes that deadlines are given [13, 20]. Transactions in certain real-time database (RTDB) applications, such as traffic monitoring and target tracking, may have deadlines determined by the nature of the application. However, in an open system such as an e-commerce system, users may simply assign arbitrarily short deadlines to their data service requests, if they are allowed to do it. Thus, an effective approach for a RTDB system to assign feasible deadlines to incoming data service requests is highly desired.

To address the problem, we develop a **fine-grained deadline assignment scheme** that takes the database schema and estimated transaction sizes into account to derive feasible deadlines for real-time data service requests in an open environment. Essentially, shorter deadlines are assigned to smaller transactions expected to access a smaller number of data items and vice versa. After assigning deadlines to data service requests by considering estimated transaction sizes, we apply the preemptive EDF algorithm to schedule them. As a result, a small request that arrived early will be served in a preferable manner in terms of

scheduling. Thus, our approach is aware of timing constraints and fairer than most of existing non-real-time databases that do not apply real-time scheduling techniques. For example, if transactions are associated with no deadlines and scheduled in a FIFO manner in a non-real-time database, a short transaction can be blocked for a long time due to long transactions that simply arrived at the system earlier. It is essential to assign feasible deadlines to real-time data service requests by considering their individual data needs. However, related work is scarce [13, 20].

To support timing constraints for real-time data services even in the presence of dynamic workloads, we apply control theoretic techniques [8]. Especially, via feedback control, we aim to support the **desired tardiness** that is expressed as the ratio of the actual service delays to the transaction deadlines. We take this approach due to the expressiveness of the tardiness metric especially for soft real-time systems; it increases as the load increases and vice versa. It is known that feedback control of the deadline miss ratio is very hard [17]. The miss ratio is 0 when the system is not overloaded, but it abruptly increases under overload. As a result, feedback-based miss ratio controller could become unstable. Due to the reason, a number of the existing feedback-based methods for real-time performance management, including [23, 3, 2], intend to *indirectly* meet deadlines by keeping the CPU utilization below the schedulable CPU utilization bound, e.g., 0.69 in the rate monotonic algorithm [14], even given dynamic workloads. However, the utilization also saturates at 1. In summary, the the utilization and miss ratio saturate at 1 and 0 when the system is overloaded and underutilized, incurring an asymmetry problem and instability [5].

In contrast, the tardiness indicates how close to the deadlines data service requests are completed in a fine-grained manner, rather than simply stating how many of the transaction deadlines are met or indirectly indicating the utilization. Thus, the tardiness metric is more suitable to design a robust closed-loop approach for real-time data services. Despite the importance, related work on feedback control of the tardiness of real-time data services is relatively scarce. Recent work on QoS-aware data services [10, 9] supports a system-wide single response time bound for all transactions via feedback control. Although this approach can effectively manage the overall response time in a database system, it is gross-grained and only provides limited real-time support, as all transactions receive the same response time bound regardless of their individual data access needs.

For performance evaluation, we have conducted extensive experiments in our stock-trading testbed that consists of the BerkeleyDB [6]−a popular open-source state-of-art database−plus a stock quote server and client threads running on Linux machines. Especially, in this paper, we extend the BerkeleyDB to support fine-grained deadline assignment and EDF scheduling. In the experiments, our closed-loop approach closely supports the desired average and transient tardiness for dynamic workloads. Moreover, compared to the BerkeleyDB extended with EDF support, our approach significantly reduces the average/transient service delay and deadline miss ratio by effectively controlling the tardiness in the closed-loop, while considerably improving the timely throughput, i.e., the total number of data accessed within the data service deadlines. Note that our deadline assignment scheme and tardiness control approach are lightweight; they only take approximately 2% CPU utilization and less than 5KB of memory in total.

The remainder of this paper is organized as follows. The overall structure of our database system is described in Section 2. Our deadline assignment method is discussed in Section 3. A description of feedback-based tardiness control is given in Section 4. Performance evaluation results are described in Section 5. Section 6 discusses related work. Finally, Section 7 concludes the paper and discusses future work.

2 Real-Time Data Service Objectives and System Overview

In this section, the objective of our QoS-aware data services and the overall system architecture are discussed.

2.1 QoS-Aware Data Service Objectives

In this paper, the tardiness t_i of the i^{th} data service request is defined as:

$$t_i = \frac{s_i}{r_i} \tag{1}$$

where s_i is the actual delay for servicing the i^{th} data service request and r_i is the relative deadline of the request. The service delay s_i is the sum of the TCP connection delay, queuing delay in the ready queue in Figure 1, and processing delay inside the database. In Table 1, an exemplar service level agreement (SLA) considered in this paper is shown. In this SLA, the tardiness set-point S_t is set to 0.8 or 0.9. In our performance evaluation presented in Section 5, we use the two tardiness set-points to observe potential trade-offs

Table 1. Desired Performance

Notation	Description	Desired Value
S_t	Tardiness Set-point	0.8 or 0.9
S_v	Tardiness Overshoot	$1.02S_t$
T_v	Settling Time	$12s$

between the different set-points in terms of the service delay, deadline miss ratio, and timely database throughput.

In Table 1, we also specify the transient performance requirements. An overshoot S_v, if any, is a transient tardiness larger than S_t. In this paper, it is desired that $S_v \leq 1.02S_t$. Also, it is desired for an overshoot to become equal to or less than S_t within the settling time $T_v = 12s$.

For feedback control, the $k^{th}(\geq 1)$ sampling period is the time interval $[(k-1)P, kP)$ and the k^{th} sampling point is equal to time kP. In this paper, we set the sampling period $P = 1s$. As hundreds of (or more) transactions finish in 1s in our stock trading testbed, performance measurement for $P = 1s$ is reliable.

2.2 System Architecture

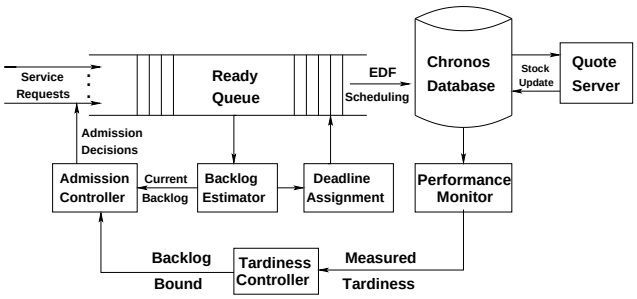


Figure 1. System Architecture

Figure 1 shows the architecture of our real-time data service testbed, called Chronos, built on top of Berkeley DB [6]. The database server processes data service requests and periodically updates stock prices received from the stock quote server to support the freshness of stock prices. 3000 stock prices are periodically updated. The update period is in a range of $[0.2s, 5s]$ for each stock. In this paper, the relative deadline of a temporal data update is equal to its update period. We consider periodic temporal data updates, since periodic updates are common in RTDBs to support data temporal consistency [20, 24]. In Chronos, we reserve dedicated threads to update stock prices. These threads for periodic updates are always executed to support the data freshness. Admission control is only applied to user requests, if necessary, to avoid overload.

We derive explicit deadlines for service requests based on the individual transaction size estimated by the backlog estimator. (A more detailed discussion is given in Section 3.) The database server schedules the accepted requests via the preemptive EDF scheduling algorithm implemented using the POSIX SCHED_FIFO scheduling class. For concurrency control, we currently apply 2PL (two phase locking). Implementing a real-time concurrency control protocol such as 2PL-HP [1] is reserved for future work. As most existing databases support 2PL and the POSIX standard is widely supported, our approach is easy to deploy.

When a new service request arrives at the system, it has to pass an admission test based on the control signal computed by the feedback-based tardiness controller (discussed in Section 4). Let $c(k)$ be the number of the data service requests, i.e., transactions and queries, processed in the k^{th} sampling period. Our feedback-based tardiness controller works as follows:

1. At the k^{th} sampling point, the performance monitor in Figure 1 computes the tardiness $t(k) = \sum_{i=1}^{c(k)} t_i/c(k)$ and tardiness error $e(k) = S_t - t(k)$. Based on $e(k)$, the tardiness controller computes the required database load bound adjustment $\delta\ell(k)$.

2. Using $\delta\ell(k)$ derived by the tardiness controller at the k^{th} sampling point, the admission controller updates the database load bound to be used in the $(k+1)^{th}$ sampling period:

$$\ell(k) = \ell(k-1) + \delta\ell(k) \tag{2}$$

   where the initial control signal $\ell(0)$ is a small positive integer at the system start.

3. In the $(k+1)^{th}$ sampling period, an incoming transaction is accepted, if the estimated backlog in the ready queue in Figure 1 does not exceed $\ell(k)$ after accepting the new request. Otherwise, the database drops the request and returns a busy message to the corresponding client.

3 Deadline Assignment for Real-Time Data Services

To derive individual deadlines for real-time data services, we first measure the average delay for accessing a single data item, u. The average delay for accessing a single data item is estimated by measuring the delay

for accessing a large number N (> 60000 in this paper) of service requests:

$$u = \frac{\sum_{i=1}^{N} s_i}{\sum_{i=1}^{N} n_i} \quad (3)$$

where s_i and n_i are the processing delay and size of transaction i, respectively. In this paper, u is derived during system identification discussed in Section 4.

For the i^{th} data service request, we estimate the number of data accesses, n_i, needed to process the request by parsing the data service request. Using n_i and the average delay for single data access, we compute the estimated delay for processing the i^{th} data service request, σ_i, as follows:

$$\sigma_i = u \times n_i. \quad (4)$$

Chronos provides four types of transactions: view-stock, view-portfolio, purchase, and sale for seven tables [10], similar to the TPC-W database benchmark for e-commerce [22]. In addition, we support periodic temporal data updates for real-time data services unlike TPC-W. Note that we do not estimate the number of data accessed by temporal data updates, because each temporal data is always updated at its update period via resource reservation as discussed in Section 2. To estimate the size of a service request, we leverage the database schema information usually available in a database system and semantics of transactions and queries as follows.

- *View-Stock*: To process this query, Chronos needs to access STOCKS and QUOTES tables that hold <stock symbol, full company name, company ID>[1]. By parsing the query, Chronos finds the number of companies n_c specified in the query. Chronos then calculates the number of data to access for T_i: $n_i = n_c \cdot \{r(\text{STOCKS}) + r(\text{QUOTES})\}$ where $r(x)$ is the average size of a row (i.e., the average number of bytes in a row) in table x.

- *View-Portfolio*: A client issues this query to see certain stock prices in its portfolio. For each stock item in the portfolio, Chronos looks up the PORTFOLIOS table that holds <client ID, company ID, purchase price, shares> to find the company IDs used to look up the QUOTES table. Thus, the estimated amount of data accessed by this query is: $n_i = |portfolio(id)| \cdot \{r(\text{PORTFOLIOS}) + r(\text{QUOTES})\}$ where $|portfolio(id)|$ is the number of stock items in the portfolio owned by the client whose ID is id.

- *Purchase*: If a client places a purchase order for a stock item, Chronos first retrieves the current stock price from the QUOTES table. If the purchased stock item was not in the portfolio before the purchase, the stock item and its purchase price are appended to the portfolio. If it is already in the portfolio, Chronos updates the corresponding shares. Hence, the estimated amount of data accessed by a PURCHASE transaction is: $n_i = r(\text{QUOTES}) + (|portfolio(id)| + 1) \cdot r(\text{PORTFOLIOS})$.

- *Sale*: To process a sale transaction, Chronos scans the PORTFOLIOS table to look up the stock items belonging to this client's portfolio. Using the stock IDs found in the PORTFOLIOS table, Chronos searches the QUOTES table to find the corresponding stock prices. After that, Chronos updates the client's portfolio in the PORTFOLIOS table to indicate the sale. Thus, the estimated amount of data accessed by a SALE transaction is: $n_i = |portfolio(id)| \cdot r(\text{PORTFOLIOS}) + n_{sell} \cdot r(\text{QUOTES}) + n_{sell} \cdot r(\text{PORTFOLIOS})$ where n_{sell} is the number of stock items to sell.

The effectiveness of the transaction size estimation technique is shown in our previous work [10]. In this paper, we consider the problems of assigning deadlines to individual transactions based on their data needs and closed-loop tardiness control for real-time data services that have not been considered in [10].

Each incoming service request is associated with a slack factor that the database system uses to control the tightness of the deadline with respect to the estimated processing time σ_i. The relative deadline of service request, r_i, is calculated as follows:

$$r_i = f_i \times \sigma_i \quad (5)$$

where f_i (≥ 1) is the slack factor and the estimated processing delay σ_i is calculated in Eq 4. The absolute deadline used in EDF scheduling is calculated by adding the request arrival time to the relative deadline r_i.

4 Design of the Tardiness Controller

We apply feedback control theory [8] to support the desired tardiness bound in the presence of dynamic workloads. To model the relation between the database

[1] In fact, the schema of Chronos is more complex with more attributes needed for online stock trades. In this paper, we only focus on the key attributes of the schema for the clarity of presentation.

backlog and tardiness, we model the tardiness of service requests at the k^{th} sampling point via the tardiness and database backlogs measured at the previous p sampling instants. We express the relation by a difference equation in the discrete time domain:

$$t(k) = \sum_{i=1}^{p} \{a_i t(k-i) + b_i \ell(k-i)\} \quad (6)$$

where $p(\geq 1)$ is the system order [8]. Using this difference equation, we model database dynamics by considering individual transactions, potentially accessing different numbers of data.

The unknown model coefficients a_i's and b_i's in Eq 6 are derived via system identification (SYSID) [8] to minimize the sum of the squared errors of tardiness estimations based on database backlogs. As SYSID aims to identify the behavior of the controlled database system, Chronos accepts all incoming data service requests without applying admission control during SYSID. Stock prices are periodically updated as discussed before. In our SYSID procedure, 1200 client threads concurrently send data service requests to the BerkeleyDB, which is extended by deadline assignment and EDF scheduling, for one hour. Each client sends a service request and waits for the response from the database server. After receiving the transaction or query result, it waits for an inter-request time (IRT) randomly selected in a certain range before sending the next data service request. A data service request accesses 60-100 data. In this paper, a fixed slack factor of 1.5 is used to derive transaction deadlines. For SYSID, we choose the IRT range [2s, 4.5s], since the data service tardiness shows a near linear pattern in this range as shown in Figure 2. Due to our system modeling and controller tuning, the feedback controller can support the desired tardiness bound within this operating range.

To analyze the accuracy of SYSID, we use the R^2 metric [8]. A control model is acceptable, if its $R^2 \geq 0.8$ [8]. We choose the second order model that achieves a high value of $R^2 = 0.982$:

$$t(k) = 1.031t(k-1) - 0.1602t(k-2) + 0.0037\ell(k-1) - 0.0029\ell(k-2) \quad (7)$$

Although our model is only second order, our R^2 value is near 1, which is the maximum possible R^2 value, due to the expressiveness of the tardiness metric.

We derive the transfer function of the open-loop database, i.e., Berkeley DB [6] with real-time scheduling extension, by taking the z-transform [8] of Eq 7:

$$\alpha(z) = \frac{T(z)}{L(z)} = \frac{0.0037z - 0.0029}{z^2 - 1.031z + 0.1602} \quad (8)$$

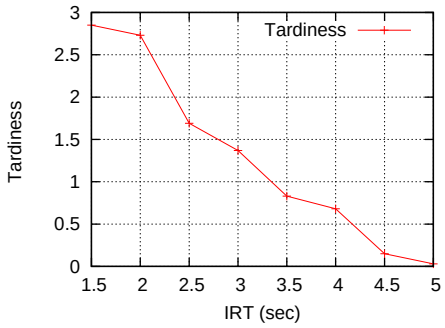


Figure 2. Inter-Request Time (IRT) vs. Tardiness

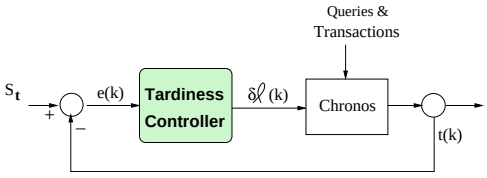


Figure 3. Tardiness Control Loop

where $T(z)$ is the z-transform of $t(k)$ and $L(z)$ is the z-transform of $\ell(k)$ in Eq 7.

Figure 3 shows the structure of the closed-loop tardiness controller. We design a PI (proportional and integral) controller to manage the tardiness. We do not use a D (derivative) controller since it is sensitive to noise. At the k^{th} sampling point, the PI controller computes the control signal $\delta\ell(k)$, i.e., the database load adjustment required to support S_t:

$$\delta\ell(k) = \delta\ell(k-1) + K_P[(K_I+1)e(k) - e(k-1)] \quad (9)$$

where the error $e(k) = S_t - t(k)$ at the k^{th} sampling point and KP and KI are proportional and integral control gains tunable to support the desired average and transient tardiness. The z-transform of Eq 9 is:

$$\beta(z) = \frac{\Delta L(z)}{E(z)} = \frac{K_P(K_I+1)[z - 1/(K_I+1)]}{z-1} \quad (10)$$

where $\Delta L(z)$ and $E(z)$ are the z-transform of $\delta\ell(k)$ and $e(k)$, respectively.

Given the open loop transfer function in Eq 8 and the transfer function of the PI controller in Eq 10, the transfer function of the closed loop in Figure 3 is: $\gamma(z) = \frac{\alpha(z)\beta(z)}{1+\alpha(z)\beta(z)}$ [8]. To support the stability of the closed-loop system and the desired performance specified in Table 1, we locate the closed loop poles−the roots of the denominator of $\gamma(z)$−inside the unit circle via the Root Locus method [8].

5 Performance Evaluation

In this section, we evaluate our approach and the baseline approach to observe whether or not they can support the desired performance specified in Table 1. We describe our experimental settings followed by the average and transient performance results.

5.1 Experimental Settings

The objective of our performance evaluation is to observe whether or not the SLA specified in Table 1 can be closely supported even in the presence of dynamic workloads. The Chronos database server runs on a desktop PC that has the dual core 3.8GHz CPU and 3GB memory. The quotes update server has the dual core 3.0GHz CPU and 2GB memory. The clients run in a machine that has the 2.5GHz CPU and 512M memory. Every machine runs the 2.6.28 Linux kernel.

For 80% of time, a client thread issues a query about stock prices. For the remaining 20% of time, a client requests a portfolio browsing, purchase, or sale transaction at a time. In fact, most service requests in e-commerce are queries. Thus, this setting is realistic. One experiment runs for 600s. For experimental purposes, we model dynamic workloads. At the beginning of one experiment, the IRT is randomly distributed in [4s, 4.5s]. At 200s, the range of the IRT is suddenly reduced to [1.5s, 2s] to model bursty workload changes, and stays in the new range until the end of the experiment. Therefore, at 200s, we abruptly increase the load by $2-3$ times to model bursty request arrivals that may create overload conditions.

For performance comparisons, we consider the following approaches:

- **RT-BDB** is the Berkeley DB [6] extended with basic real-time support. Individual transaction deadlines are derived according to our deadline assignment scheme presented in Section 3. We apply the preemptive EDF algorithm to schedule incoming service requests. However, *RT-BDB* does not have the closed-loop tardiness control system and accepts all incoming requests. Hence, it provides basic real-time features similar to the ones provided by state-of-art (proprietary) RTDBs, such as [15, 19].

- **TC** is the closed-loop tardiness control system designed in this paper to support real-time data service requirements. It also applies the deadline assignment scheme and preemptive EDF scheduling. To thoroughly evaluate the effectiveness of our approach, we use two different tardiness set-points to observe their impacts on the performance and potential trade-offs. Our closed-loop approach with different set-points, 0.9 and 0.8, are denoted as *TC-SP1* and *TC-SP2*, respectively.

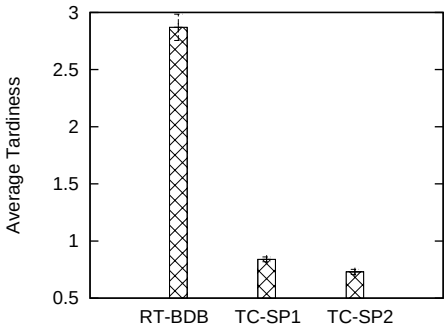


Figure 4. Average Tardiness

Note that we only compare the performance of our approach to *RT-BDB*, because most of the previous work on feedback-based real-time data services, including [4, 12, 11], assumes transaction deadlines are already given without considering database-specific deadline assignment issues. Neither do they consider tardiness control. On the other hand, more recent work on feedback control of data service delays including [10, 9] does not consider individual transaction deadlines but a single response time bound. Therefore, it is impossible for us to directly compare our approach to them. Also, it is well known that feedback-based systematic admission control significantly outperforms static or ad-hoc heuristic-based approaches for admission control [17, 10].

Each performance data is the average of 10 runs using different random seeds. We show the performance results observed between $100s$ and $600s$ to exclude the database initialization phase, which involves initial housekeeping chores such as database schema and data structure initialization.

5.2 Average Performance Analysis

Figure 4 shows the average tardiness for tested approaches. *RT-BDB* shows the average tardiness value of 2.87; that is, the service delay is 2.87 times the transaction deadline in average. In contrast, *TC-SP1* and *TC-SP2* are able to support the corresponding average tardiness bound, 0.9 and 0.8, respectively.

Figure 5 depicts the average deadline miss ratio. *RT-BDB* shows the approximately 86% (deadline) miss ratio. However, *TC-SP1* and *TC-SP2* support the 4.3% and 1.9% miss ratios by systematically controlling the tardiness of service requests in the closed-loop.

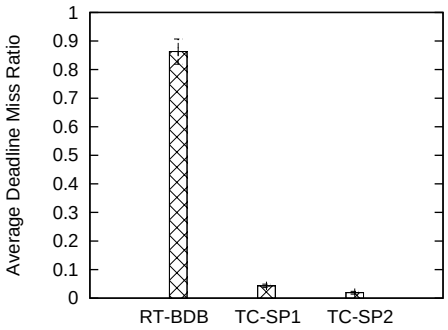


Figure 5. Average Deadline Miss Ratio

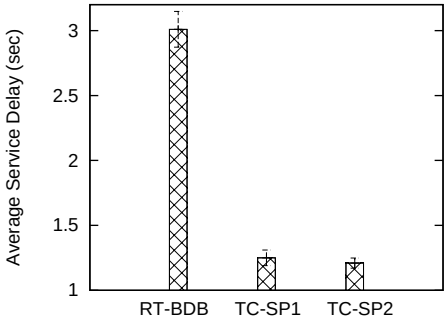


Figure 6. Average Service Delay

Figure 6 shows that *TC-SP1* and *TC-SP2* support the substantially shorter average service delay compared to *RT-BDB*. As shown in Figure 7, *TC-SP1* and *TC-SP2* increase the total number of data processed per second by timely transactions that finish within their deadlines by up to an order of magnitude compared to *RT-BDB*.

Using Figures 4-7, we also compare the relative performance of *TC-SP1* and *TC-SP2*. We observe that *TC-SP2*'s miss ratio is smaller than *TC-SP1*'s miss ratio, because *TC-SP2* supports a smaller tardiness set-point, which demands more stringent timing requirements, at the cost of the reduced timely throughput compared to *TC-SP1*. Hence, an application administrator can choose a smaller tardiness set-point, if timeliness is more important than the timely throughput and vice versa.

5.3 Transient Performance Analysis

Figures 8, 9 and 10 show the transient performance of tested approaches at every 1s sampling period. As shown in Figure 8, *RT-BDB* largely fails to support the transient performance goals specified in Table 1. It shows excessive tardiness overshoots when the workload increases at 200s and the tardiness overshoots

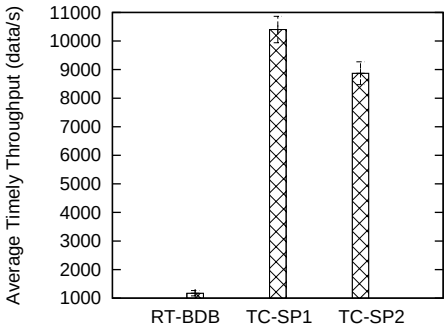


Figure 7. Average Timely Throughput

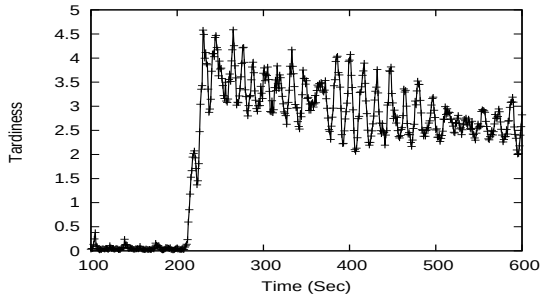
range between 2 and 4.5. As a result, it suffers significant performance degradation in terms of the transient miss ratio, service delay and timely throughput after 200s as shown in Figure 8.

In contrast, our tardiness control approach closely supports the transient tardiness bound. As shown in Figures 8 and 9, *TC-SP1* and *TC-SP2* only show small magnitudes of overshoots after the abrupt workload change at 200s and all the overshoots reduce to be below the set-point within 12s, i.e., the desired settling time specified in Table 1. These results show the effectiveness of tardiness control. From Figures 8, 9 and 10, we also observe that our approach significantly reduces the transient deadline miss ratio and service delay, while substantially improving the transient timely throughput compared to *RT-BDB*.

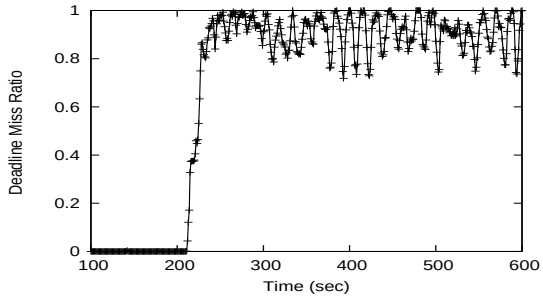
6 Related Work

Various aspects of real-time data management has been studied [13, 20]. However, relatively little work has been done to investigate the problem of deadline assignment to user transactions in an open environment. Xiong et al. [25, 24] investigate the problem of deadline assignment to *update transactions* scheduled by the deadline monotonic and EDF scheduling algorithms in RTDBs. They design novel approaches to judicially assign periods and deadlines to temporal data updates to reduce the update workload. However, they do not consider the problem of assigning deadlines to user data service requests, which is the focus of our work. Also, feedback-based tardiness control is not considered in their work. Thus, our work is complementary to their work.

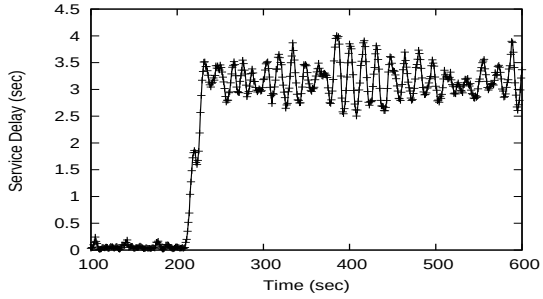
Feedback control theory has been applied to support real-time data services in the presence of dynamic workloads [3, 4, 11]. However, they do not consider the deadline assignment problem or tardiness metric for feedback control of real-time data service perfor-



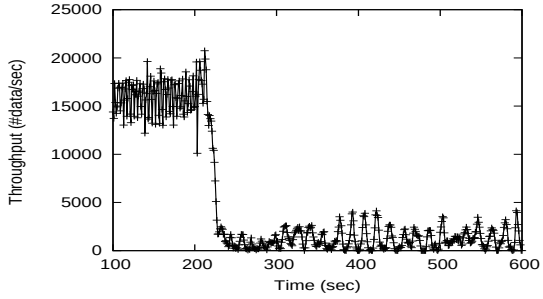
(a) Tardiness



(b) Deadline Miss Ratio

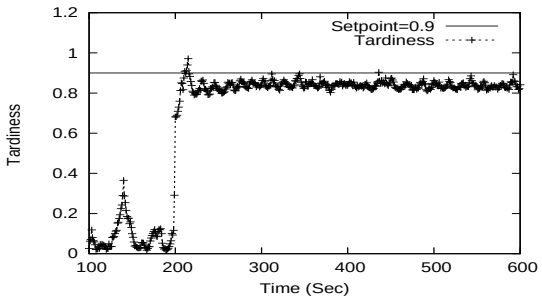


(c) Service Delay

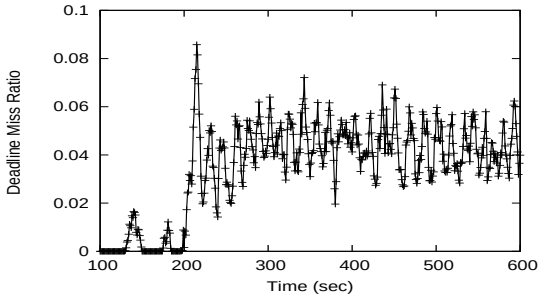


(d) Timely Throughput

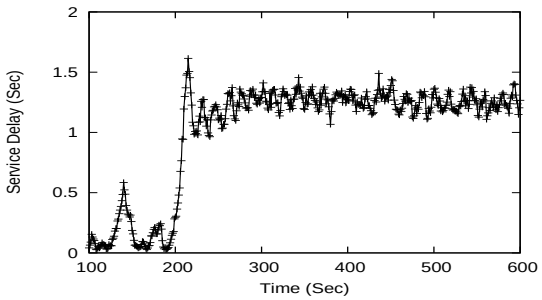
Figure 8. Transient Performance of *RT-BDB*



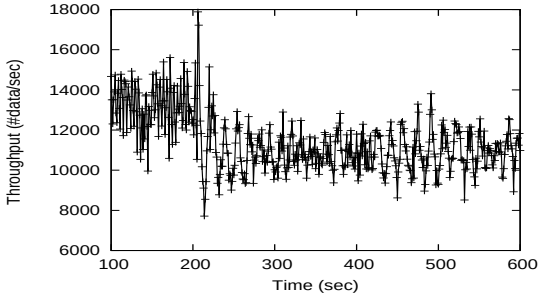
(a) Tardiness



(b) Deadline Miss Ratio

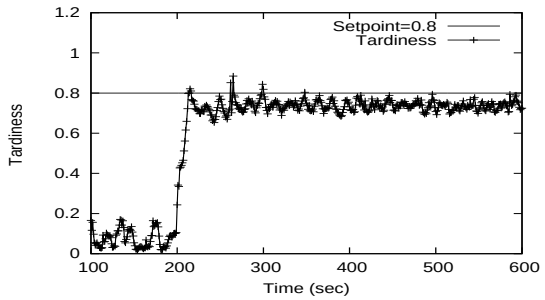


(c) Service Delay

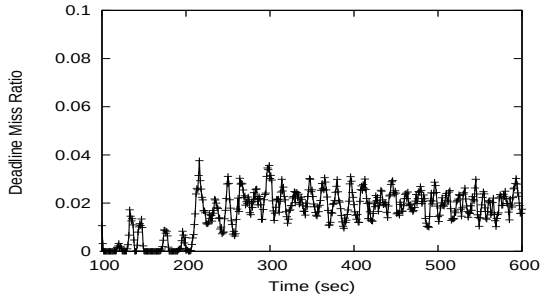


(d) Timely Throughput

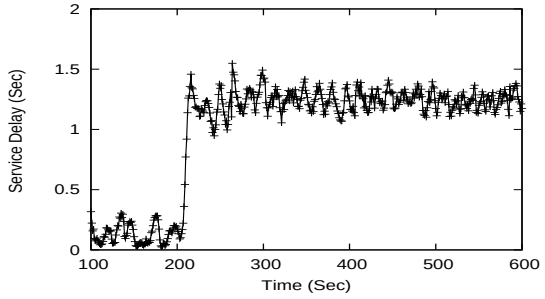
Figure 9. Transient Performance of *TC-SP1* **(Set-point = 0.9)**



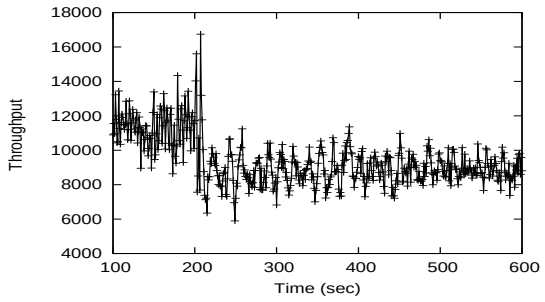
(a) Tardiness



(b) Deadline Miss Ratio



(c) Service Delay



(d) Timely Throughput

Figure 10. Transient Performance of *TC-SP2* (Set-point = 0.8)

mance. Neither are these approaches implemented and evaluated in a real database system unlike our work presented in this paper. Our recent work on QoS-aware data services [10, 9] aims to support the average response time bound in a real database system. However, individual transaction deadlines and tardiness control are not considered. QeDB [12] aims to support the QoS in terms of the average tardiness of the transactions in a real-time embedded database. The key idea is to divide the transaction deadline into CPU deadline and I/O deadline and dynamically adjust the I/O and CPU deadlines at run-time based on the system resource usage. In QeDB, transactions are classified into three classes and they are scheduled in a FIFO manner in each class rather than applying real-time scheduling techniques. Also, it does not consider the problem of deadline assignment based on individual transaction needs in an open environment. Thus, our deadline assignment and tardiness control scheme are different from theirs. Control theory has been applied to manage the performance of computer systems such as a web server [16, 18]. However, they do not consider RTDB-specific issues such as transaction deadline assignment and tardiness control based on the derived transaction deadlines.

In [17], the authors apply feedback control techniques to support the target deadline miss ratio by manipulating the system load. However, the miss ratio controller shows relatively unstable performance, since the miss ratio is saturated at 0 when the system is not overloaded and it abruptly increases under overload as discussed before. Other feedback-based approaches, including [23, 3, 2], aim to indirectly control the CPU utilization to meet deadlines. Bertini et al. [5] show that only counting the number of missed deadlines per sampling period results in poor accuracy and large confidence intervals in terms of measuring and controlling the performance of web services. They define a novel QoS metric, named tardiness quantile metric, and design a feedback control scheme to support the user specified tardiness quantile. However, these approaches do not consider the data needs of individual real-time transactions to assign feasible deadlines and accordingly design a closed-loop system to support the desired average/transient tardiness of real-time data services. Block et al. [7] developed a feedback-based scheduling framework for adapting the processor shares of tasks on real-time multiprocessor platforms. In the future, we will investigate the performance of real-time data services on multiprocessor platforms.

7 Conclusions and Future Work

It is challenging to support real-time data services in data-intensive real-time applications, e.g., e-commerce, due to dynamic workloads. This paper tackles the challenge as follows: 1) We develop a fine-grained deadline assignment method to derive individual deadlines of real-time data service requests based on their data needs; 2) We design a tardiness control model for real-time data services and develop the tardiness controller; 3) Performance is thoroughly evaluated in a stock trading testbed. Our approach supports the desired average/transient tardiness, while significantly outperforming the BerkeleyDB with real-time extensions in terms of the miss ratio and timely throughput. Despite the importance, relatively little work has been done for deadline assignment and tardiness control to support real-time data services. In the future, we will continue to investigate more efficient approaches for real-time data services.

References

[1] R. Abbott and H. Garcia-Molina. Scheduling Real-Time Transactions: A Performance Evaluation. *ACM Transactions on Database System*, 17:513–560, 1992.

[2] M. Amirijoo, N. Chaufette, J. Hansson, S. H. Son, and S. Gunnarsson. Generalized Performance Management of Multi Class Real-Time Imprecise Data Services. In *the 26th IEEE Real-Time Systems Symposium*, 2005.

[3] M. Amirijoo, J. Hansson, S. Gunnarsson, and S. H. Son. Experimental Evaluation of Linear Time-Invariant Models for Feedback Performance Control in Real-Time System. *Real-Time Systems*, 35(3):209–238, 2007.

[4] M. Amirijoo, J. Hansson, and S. H. Son. Specification and Management of QoS in Real-Time Databases Supporting Imprecise Computations. *IEEE Transactions on Computers*, 55(3):304–319, 2006.

[5] L. Bertini, J. Leite, and D. Mossé. Statistical QoS Guarantee and Energy-Efficiency in Web Server Clusters. In *the 19th Euromicro Conference on Real-Time Systems*, 2007.

[6] Oracle Berkeley DB Product Family. Available at http://www.oracle.com/database/berkeley-db/index.html.

[7] A. Block, B. Brandenburg, J. Anderson, and S. Quint. An Adaptive Framework for Multiprocessor Real-Time Systems. In *the 20th Euromicro Conference on Real-Time Systems*, 2008.

[8] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. A John Wiley and Sons, Inc., Publication, 2004.

[9] K. D. Kang, J. Oh, and S. H. Son. Chronos: Feedback Control of a Real Database System Performance. In *IEEE Real-Time Systems Symposium*, 2007.

[10] K. D. Kang, J. Oh, and Y. Zhou. Backlog Estimation and Management for Real-Time Data Services. In *the 20th Euromicro Conference on Real-Time Systems*, 2008.

[11] K. D. Kang, S. H. Son, and J. A. Stankovic. Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases. *IEEE Transactions on Knowledge and Data Engineering*, 16(10):1200–1216, 2004.

[12] W. Kang, S. H. Son, and J. A. Stankovic. QeDB: A Quality-Aware Embedded Real-Time Database. In *the 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2009.

[13] K. Y. Lam and T. W. Kuo, editors. *Real-Time Database Systems*. Kluwer Academic Publishers, 2006.

[14] J. W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[15] Lockheed Martin. EagleSpeed Real-Time Database Manager.

[16] C. Lu, Y. Lu, T. Abdelzaher, J. Stankovic, and S. Son. Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers. *IEEE Transactions on Parallel and Distributed Systems*, 17(9), Sept. 2006.

[17] C. Lu, J. Stankovic, G. Tao, and S. Son. Design and Evaluation of a Feedback Control EDF Algorithm. In *Real-Time Systems Symposium*, December 1999.

[18] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using Control Theory to Achieve Service Level Objectives in Performance Management. *Real-Time Systems*, 23(1-2):127–141, 2002.

[19] Polyhedra. Polyhedra White Papers. http://support.polyhedra.com/embedded_database.htm.

[20] K. Ramamritham, S. H. Son, and L. C. Dipippo. Real-Time Databases and Data Services. *Real-Time Systems Journal*, 28, Nov.-Dec. 2004.

[21] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo. *Deadline scheduling for real-time systems*. Kluwer Academic Publishers, 1998.

[22] Transaction processing performance council. http://www.tpc.org/.

[23] X. Wang, D. Jia, C. Lu, and X. Koutsoukos. DEUCON: Decentralized End-to-End Utilization Control for Distributed Real-Time Systems. *IEEE Transactions on Parallel and Distributed Systems*, 18(7):996–1009, 2007.

[24] M. Xiong and K. Ramamritham. Deriving Deadlines and Periods for Real-Time Update Transactions. *IEEE Transactions on Computers*, 53(5):567–583, 2004.

[25] M. Xiong, Q. Wang, and K. Ramamritham. On Earliest Deadline First Scheduling for Temporal Consistency Maintenance. *Real-Time System*, 40(2):208 – 237, 2008.